19 | 设计模式（上）：C++与设计模式有啥关系?

2020-06-18 罗剑锋

讲述：Chrono
时长 13:40 大小 12.53M

你好，我是 Chrono。

今天，我们进入最后的“总结”单元，把前面学到的这些知识上升到“理论结合实践”的高度，做个归纳整理。我们先来了解一下设计模式和设计原则，然后再把理论“落地”，综合利用所有知识点，设计并开发出一个实际的服务器应用。

你可能会问了：我们这是个 C++ 的课程，为什么还要专门来讲设计模式呢?

我觉得，设计模式是一门通用的技术，是指导软件开发的“金科玉律”，它不仅渗透进 C++ 语言和库的设计（当然也包括其他编程语言），而且也是成为高效 C++ 程序员必不可缺的“心法”和“武器”。

掌握了它们，理解了语言特性、库和工具后面的设计思想，你就可以做到“知其然，更知其所以然”，然后以好的用法为榜样，以坏的用法为警示，扬长避短，从而更好地运用 C++。

所以，我把我这些年的实践经验进行了提炼和总结，糅合成了两节课，帮你快速掌握，并且用好设计模式，写出高效、易维护的代码。这节课，我会先讲一讲学好设计模式的核心方法，下节课，我们再讲在 C++ 里具体应用了哪些设计模式。

为什么要有设计模式?

虽然 C++ 支持多范式编程，但面向对象毕竟还是它的根基，而且，面向对象编程也通用于当前各种主流的编程语言。所以，学好、用好面向对象，对于学好 C++ 来说，非常有用。

但是，想要得到良好的面向对象设计，并不是一件容易的事情。

因为每个人自身的能力、所在的层次、看问题的角度都不同，仅凭直觉“对现实建模”，很有可能会生成一些大小不均、职责不清、关系混乱的对象，最后搭建出一个虽然可以运行，但却难以理解、难以维护的系统。

所以，设计模式就是为此而生的。

它系统地描述了一些软件开发中的常见问题、应用场景和对应的解决方案，给出了**专家级别的设计思路和指导原则**。

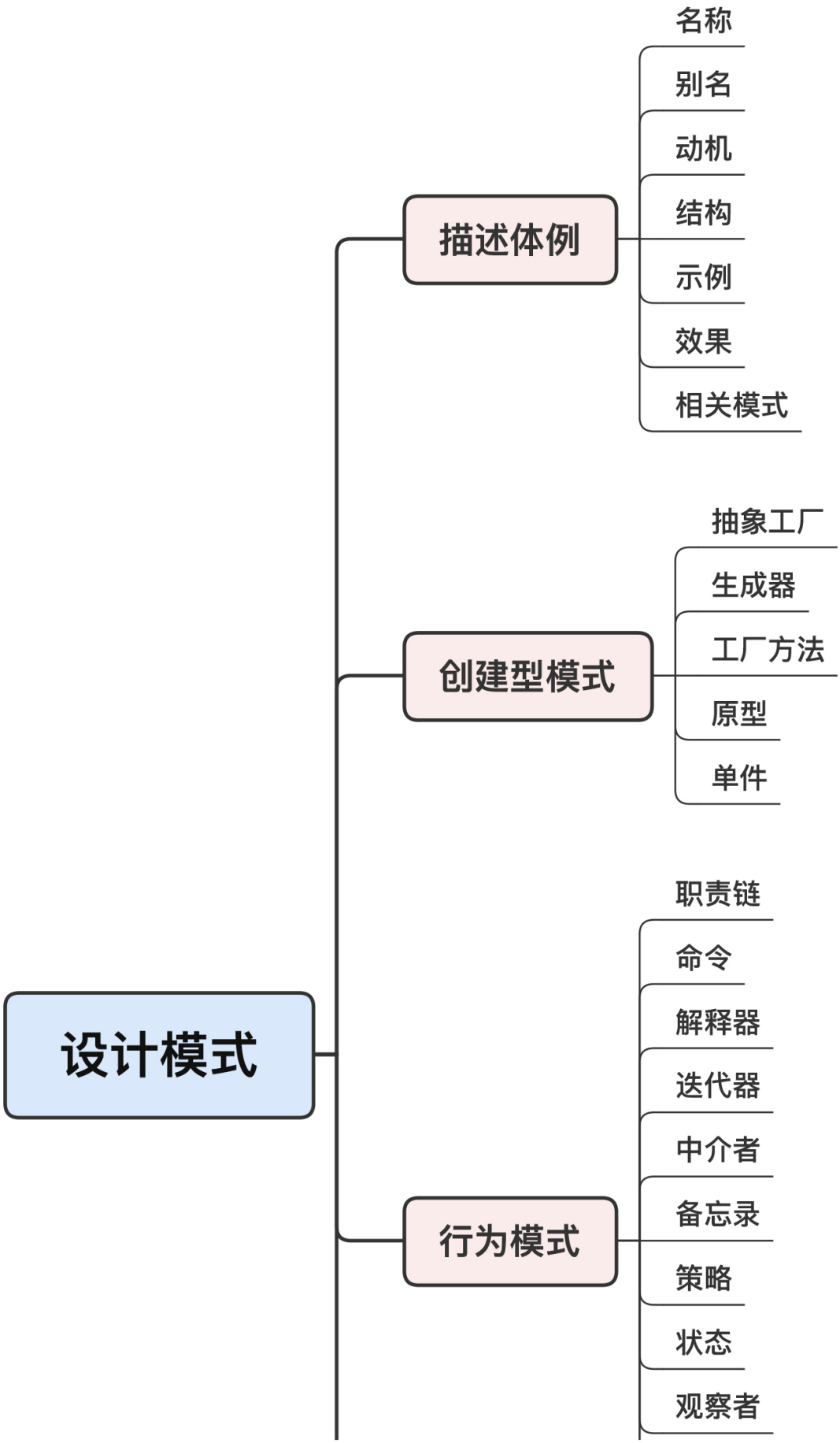
按照设计模式去创建面向对象的系统，就像是由专家来“手把手”教你，不能说绝对是“最优解”，但至少是“次优解”。

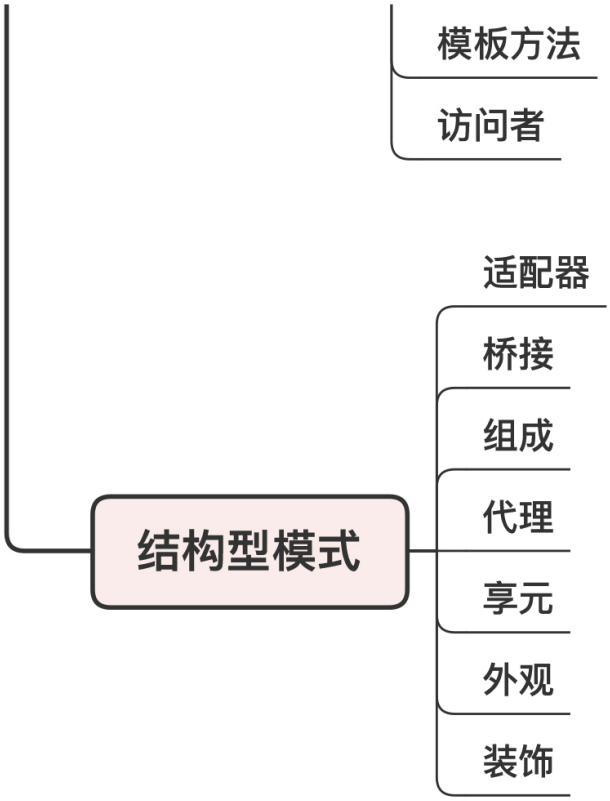
而且，在应用设计模式的过程中，你还可以从中亲身体会这些经过实际证明的成功经验，潜移默化地影响你自己思考问题的方式，从长远来看，学习和应用设计模式能够提高你的面向对象设计水平。

学习、理解设计模式，才能用好面向对象的 C++

经典的《设计模式》一书里面介绍了 23 个模式，并依据设计目的把它们分成了三大类：创建型模式、结构型模式和行为模式。

这三类模式分别对应了开发面向对象系统的三个关键问题：**如何创建对象、如何组合对象，以及如何处理对象之间的动态通信和职责分配**。解决了这三大问题，软件系统的“架子”也就基本上搭出来了。





极客时间

23 个模式看起来好像不是很多，但它们的内涵和外延都是非常丰富的，不然也不会有数不清的论文、书刊研究它们了，所以，我们要从多角度、多方面去评价、审视模式。

那该怎么做才好呢？

你可以看一下《设计模式》的原书，它用了一个很全面的体例来描述模式，包括名称、别名、动机、结构、示例、效果、相关模式，等等。

虽然显得有点琐碎、啰唆，但我们必须要承认，这种严谨、甚至是有些刻板的方式能够全方位、无死角地介绍模式，强迫你从横向、纵向、深层、浅层、抽象、具体等各个角度来研究、思考。只有在这个过程中，你才能真正掌握设计模式的内核。

模式里的结构和实现方式直接表现为代码，可能是最容易学习的部分，但我认为，其实这些反而是最不重要的。

你更应该去关注它的参与者、设计意图、面对的问题、应用的场合、后续的效果等代码之外的部分，它们通常比实现代码更重要。

因为代码是“死”的，只能限定由某几种语言实现，而模式发现问题、分析问题、解决问题的思路是“活”的，适用性更广泛，这种思考“What、Where、When、Why、How”并逐步得出结论的过程，才是设计模式专家经验的真正价值。

理解了这些内容，我们就可以应用在 C++ 面向对象编程里了。下节课，我会具体给你讲一讲在 C++ 里，这些该怎么用。

学习、理解设计原则，才能用好多范式的 C++

可能你在学习设计模式的时候还是有些困惑，设计模式是专家经验的总结不假，但专家们是如何察觉、发现、探索出这些模式的呢？

而且模式真的完全只是“模式”、固定的“套路”，有没有什么更一般的思想来指导我们呢？换句话说，有没有“设计‘设计模式’的模式”呢？

嗯，这个真的有（笑）。

其实，这些更高层次的指导思想你可能也听说过，它们被通称为“设计原则”。

最常用有 5 个原则，也就是常说的“SOLID”。

1. SRP，单一职责（Single ResponsibilityPrinciple）；
2. OCP，开闭（Open Closed Principle）；
3. LSP，里氏替换（Liskov Substitution Principle）；
4. ISP，接口隔离（Interface-Segregation Principle）；
5. DIP，依赖反转，有的时候也叫依赖倒置（Dependency Inversion Principle）。

不过可能是因为我最先接触、研究的是设计模式，所以后来再看到这些原则的时候，“认同感”就没有那么强烈了。

虽然它们都说得很对，但没有像设计模式那样给出完整、准确的论述。所以，我觉得它们有点“飘”，缺乏可操作性，在实践中不好把握使用的方式。

但另一方面，这些原则也确实提炼出了软件设计里最本质、最基本的东西，就好像是欧几里得五公设、牛顿三定律一样，初看上去似乎很浅显直白，但仔细品品，就会发现，可以应用到任何系统里，所以了解它们还是很有必要的。

下面我就来讲讲对设计原则的一些理解和看法，再结合 C++ 和设计模式，帮你来加深认识，进而在 C++ 里实际用好它们。

第一个，**单一职责原则**，简单来说就是“**不要做多余的事**”，更常见的说法就是“**高内聚低耦合**”。在设计类的时候，要尽量缩小“粒度”，功能明确单一，不要设计出“大而全”的类。

使用单一职责原则，经常会得到很多“短小精悍”的对象，这时候，就需要应用设计模式来组合、复用它们了，比如，使用工厂来分类创建对象、使用适配器、装饰、代理来组合对象、使用外观来封装批量的对象。

单一职责原则的一个反例是 C++ 标准库里的字符串类 string（参见第 11 讲），它集成了字符串和字符容器的双重身份，接口复杂，让人无所适从（所以，我们应该只把它当作字符串，而把字符容器的工作交给vector<char>）。

第二个是**开闭原则**，它也许是最“模糊”的设计原则了，通常的表述是“**对扩展开放，对修改关闭**”，但没有说具体该怎么做，跟没说一样。

我觉得，你可以反过来理解这个原则，在设计类的时候问一下自己，这个类封装得是否足够好，是否可以不改变源码就能够增加新功能。如果答案是否定的（要改源码），那就说明违反了开闭原则。

应用开闭原则的关键是做好封装，隐藏内部的具体实现细节，然后开放足够的接口，这样外部的客户代码就可以只通过接口去扩展功能，而不必侵入类的内部。

你可以在一些结构型模式和行为模式里找到开闭原则的“影子”：比如桥接模式让接口保持稳定，而另一边的实现任意变化；又比如迭代器模式让集合保持稳定，改变访问集合的方式只需要变动迭代器。

C++ 语言里的 final 关键字（第 5 讲）也是实践开闭原则的“利器”，把它用在类和成员函数上，就可以有效地防止子类的修改。

第三个原则是**里氏替换原则**，意思是**子类必须能够完全替代父类**。

这个原则还是比较好理解的，就是说子类不能改变、违反父类定义的行为。像在第 5 讲里说的正方形、鸟类的例子，它们就是违反了里氏替换原则。

不过，因为 C++ 支持泛型编程，而且我也不建议多用继承，所以在 C++ 里你只要了解一下它就好。

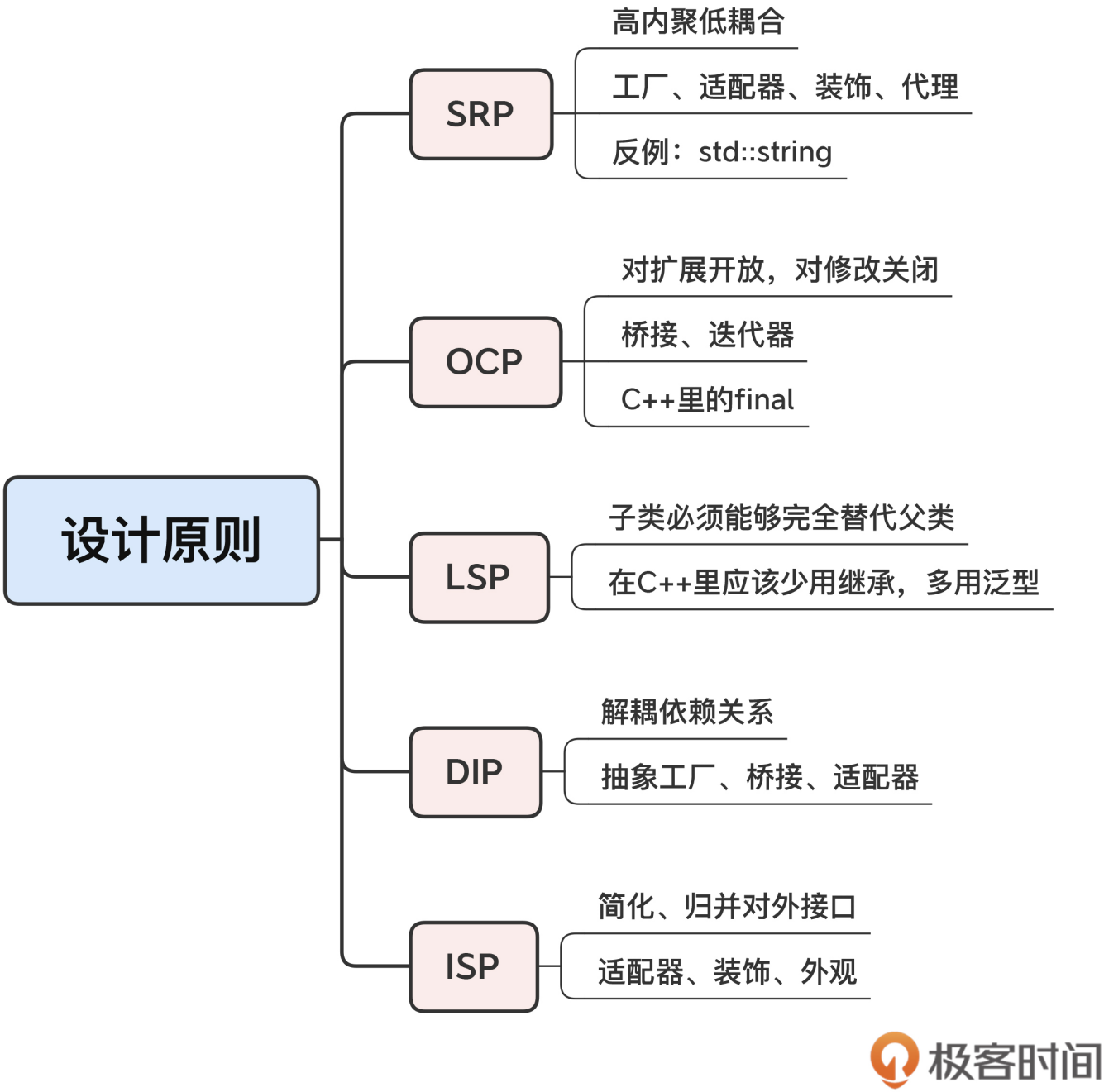
第四个是**接口隔离原则**，它和单一职责原则有点像，但侧重点是对外的接口而不是内部的功能，目标是**尽量简化、归并给外界调用的接口**，避免写出大而不当的“面条类”。

大多数结构型模式都可以用来实现接口隔离，比如，使用适配器来转换接口，使用装饰模式来增加接口，使用外观来简化复杂系统的接口。

第五个原则是**依赖反转原则**，个人觉得是一个比较难懂的原则，我的理解是**上层要避免依赖下层的实现细节，下层要反过来依赖上层的抽象定义**，说白了，大概就是“解耦”吧。

模板方法模式可以算是比较明显的依赖反转的例子，父类定义主要的操作步骤，子类必须遵照这些步骤去实现具体的功能。

如果单从“解耦”的角度来理解的话，存在上下级调用关系的设计模式都可以算成是依赖反转，比如抽象工厂、桥接、适配器。



除了 SOLID 这五个之外，我觉得还有两个比较有用：DRY（Don’t Repeate Yourself）和 KISS（Keep It Simple Stupid）。

它们的含义都是要让代码尽量保持简单、简洁，避免重复的代码，这在 C++ 里可以有很多方式去实现，比如用宏代替字面值，用 lambda 表达式就地定义函数，多使用容器、算法和第三方库。

小结

好了，今天就到这里吧，我从比较“宏观”的层面说了设计模式和设计原则。

其实这些就是对我们实际开发经验的高度浓缩和总结。理解掌握了这些经验，你就会始终保持着清醒的头脑，在写 C++ 代码的过程中有意识地去发现、应用模式，设计出好的结构，对坏的代码进行重构。

小结一下这节课的要点：

1. 面向对象是主流编程范式，使用设计模式可以比较容易地得到良好的面向对象设计；
2. 经典的设计模式有 23 个，分成三大类：创建型模式、结构型模式和行为模式；
3. 应该从多角度、多方面去研究设计模式，多关注代码之外的部分，学习解决问题的思路；
4. 设计原则是设计模式之上更高层面的指导思想，适用性强，但可操作性弱，需要多在实践中体会；
5. 最常用的五个设计原则是“SOLID”，此外，还有“DRY”和“KISS”。

不过，我还要特别提醒你，设计模式虽然很好，但它绝不是包治百病的“灵丹妙药”。如果不论什么项目都套上设计模式，就很容易导致过度设计，反而会增加复杂度，僵化系统。

对于我们 C++ 程序员来说，更是要清楚地认识到这一点，因为在 C++ 里，不仅有面向对象编程，还有泛型编程和函数式编程等其他范式，所以领会它的思想，在恰当的时候改用模板 / 泛型 /lambda 来替换“纯”面向对象，才是使用设计模式的最佳做法。

课下作业

最后是课下作业时间，给你留两个思考题：

1. 你觉得使用设计模式有什么好处？
2. 你是怎么理解 SOLID 设计原则的？哪个对你最有指导意义？

欢迎你在留言区写下你的思考和答案，如果觉得今天的内容对你有所帮助，也欢迎分享给你的朋友。我们下节课见。

课外小贴士

1. 《设计模式》一书里收录了23个设计模式，但实际存在的、有用的模式远不止这些。作者们说在成书前删去了一些，我个人怀疑这是有意为之，因为23正好是人类染色体的数量。

2. 国内对设计模式有多种不同的译法，比如Singleton的“单件”“单例”，Facade的“外观”“门面”，感觉在Java界后一种称呼比较多，这里以《设计模式》书为准。

3. 与设计模式相关的另一个概念是重构，设计模式是为了避免重构，而重构则会应用到设计模式。还有一个概念叫“反模式”，或者叫“代码坏味”，记录了编程时容易落入的“陷阱”，比如Hard code、Magic numbers。

极客时间

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　18 | 性能分析：找出程序的瓶颈

下一篇　20 | 设计模式（下）：C++是怎么应用设计模式的?

精选留言 (8)

写留言

EncodedStar
2020-06-18

设计模式是程序员的内功，学会设计模式并在实战中用到，会让你的程序变的很灵活，有种未雨绸缪的感觉。
设计模式都是前人实战留下的经典，好比三十六计一样，智者才能巧用啊。

> 作者回复: 设计模式就要多用、多思考，必须在实践中才能学到，看书看资料还是比较难领会其中的精髓的。
>
> 但有书能够让你少走弯路，快速找到好的设计，避免摸索的成本。

2

lckfa李钊

2020-06-18

编程语言只是外家功夫，练得再好也像在使蛮力，设计模式才是内功心法，算是可以让编程能力从coding到programming的良方；但是用好设计模式真的不容易，很多人告诉我用的最熟的设计模式是单例模式，这让我哭笑不得，并不是看不上单例，实在是因为单例模式并不能很好的体验设计模式的精髓。个人理解的设计模式是能脱离了代码实际，向更高维度的项目工程化进阶的利器，让我们可以更好的组织和维护代码，迭代需求。但是...

展开

作者回复: 说的非常好，里面的观点我都深有同感。

学编程，必须要学设计模式，领会它的内在思想，才能提升自己的编程功力，才能往上走。

2

Weining Cao

2020-06-18

老师设计模式总结得很好。只是有一点我觉得有点疑惑：用宏代替字面量 真的是一个好的实践吗？ 另外我理解宏定义能够减少代码量，但它也带来很不好的问题是它基本没法调试。 不知道老师有没有推荐的方法或者工具能够帮助调试C++的宏或者自动展开C++的宏变成清晰易于理解的源码？

展开

作者回复: 用宏代替字面量只是C++里的一种用法，不是特别提倡，但用起来也不错，这个完全看个人或者团队是否能够达成一致。

比如说，用C和C++混合编程，有的接口会导出成C的形式，那么用宏就很合理了。

关于宏带来的问题和它的恶名也都是众所皆知的了，我也是推荐尽量少用宏，用来定义常量还可以，要是写复杂的宏替换、宏函数就不是特别好了，会很影响可读性。

1

九三

2020-06-18

设计模式是作用在 编程范式如 函数式 面向对象 面向过程等这些范式上吗

作者回复: 严格来说，设计模式是用于面向对象的，但面向过程也可以利用抽象和封装模拟面向对象，所以设计模式也能够用在面向过程。

而函数式编程与面向对象的差异就比较大了，所以不能应用大多数模式。

1

蓝配鸡
2020-06-22

每次读老师的文字都很舒服， 就好像老师就坐在对面和你娓娓道来。 厉害👍，希望再多出些专栏

作者回复: 设计模式比较抽象，不像C++语言那么好学，有疑问、不清楚的，可以随时留言，一起讨论。

有学识的兔子
2020-06-21

我觉得 模板模式也可以算作遵循OCP的设计原则，这个模式在满足基本业务逻辑的同时，通过关键环节的扩展点，方便后续的扩展优化；
设计模式的出现是应对复杂系统问题，让复杂的系统的维护不至于超出程序员的掌控，这个角度讲，软件系统规模大到一定程度，就需要借助设计模式，让系统复杂度控制某个范围，例如某个模块，某个类等等；...
展开

作者回复: 说的很对，有的模式也会同时符合多个原则。

模式的应用范围也是可大可小，架构里可以用，代码里也可以用。

黄骏
2020-06-18

设计模式，在考高级架构师时有涉及，后面的工作中也遇到一些，比如，单例，访问者，观察者等，设计模式有点类似写作文，比如，议论文的三段法等。确实能避免很多问题

作者回复: 设计模式就是开发软件的经典“套路”，按照这个“套路”来搭建系统，就能够得到良好的设计。

但“套路”用得多了，也会渐渐地发现不足，就会想突破“套路”。

Confidant.

2020-06-18

设计模式感觉非常的玄，第一次觉得自己稍微理解了一些设计模式的皮毛的时候，是对编译链接的内容有了一些体会，发现使用设计模式，可以在编译代码的时候，只编译增量的部分，从而达到修改原来功能的效果，大概体会到设计模式中对于应对变化的一些实际效果，不过实际写代码的过程中，倒是还没有到能随时把设计模式或者这些设计思想应用到自己的代码中的功底。

展开

> 作者回复: 设计模式描述的是对象之间的关系，可能画出uml图来能够更好地帮助理解。
>
> 可以在写代码的时候多想想，如果需求、环境发生了变化，我们的代码能否不改或者少改就能适应变化，这就是设计模式发挥作用的地方。